AGENDA

- **Introduction**
  - NIC Teaming
  - RoCE and ib_device
  - Application view

- **RDMA Device HW Bonding**

- **HW Bond and virtualization**
  - Embedded Switch SW Model
  - Embedded Switch and HW Bonding

- **Multi-PCI Socket NIC**
  - Introduction
  - HW Bonding for app transparency

- **Summary**
IEEE 802.3ad defines how to combine multiple physical network ports to single logical port for:

- High Availability
- Load balancing

- Linux uses Bonding/Teaming device for building Link Aggregation trunk

- Both expose software net_dev that provides LAG I/F toward the networking stack

- Team/bond is considered “upper” device to “lower” enslaved NICs net_devices

- Different modes of operation
  - Active/Passive
  - 802.3ad (LAG) static and dynamic (LACP)

- Traditional network stack sees single “upper” net_dev
INTRODUCTION
RDMA over Ethernet (RoCE) / RDMA-CM

- The upstream RDMA stack supports multiple transports: RoCE, IB, iWARP
- RoCE – RDMA over Converged Ethernet, RoCE V2 (upstream 4.5), IBTA RDMA headers over UDP.
- RoCE uses IPv4/6 addresses set over the regular Eth NIC port net_dev
- RoCE apps use RDMA-CM API for control path and verbs API for data path
- RDMA-CM API *(include/rdma/rdma_cm.h)*
  - Address resolution – Local Route lookup + ARP/ND services (rdma_resolve_addr())
  - Route resolution – Path lookup in IB networks (rdma_resolve_route())
  - Connection establishment – per transport CM to wire the offloaded connection (rdma_connect())
- Verbs API
  - Send/RDMA – Send message or perform RDMA operation (post_send())
  - Poll– Poll for completion of Send/RDMA or Receive operation (poll_cq())
    - Async completion handling and fd semantics are supported
    - Post Receive Buffer – Hand receive buffers to the NIC (post_recv())
- RDMA Device – ib_device
  - The DEVICE structure, exposes all above operations
  - *Associated with net_device*
- Available for both RoCE and user mode Ethernet programming (e.g. DPDK)
ETHERNET BONDING
Application Point of View

User

Kernel

夔User Verbs

Kernel

HW

HCA

Ib_dev

eth0

bond0

TCP/IP

Sockets

Socket App

Verbs App

Ib_dev

eth1

Linux Bonding
RDMA DEVICE HW BONDING

- **Register new ib_dev associated with the bond net_dev**
  - eth0, eth1 will listen on Linux bond enslavement netlink events
  - New device will use provider pick of PCIe Function (PF0/1 or both) for device I/O

- **Registered RDMA devices associated with eth0, eth1**
  - Will unregister and re-register to drop existing consumers on enslavement
  - Will be used for port management only through Port Immutable ops (get_port_immutable())
    - Alike the Linux Bonding enslaved net_devs
**RDMA DEVICE HW BONDING – CONT.**

- **HW Bond**
  - NIC logic for HW forwarding of ingress traffic to bond/team RDMA device
  - `net_dev` traffic is passed directly to owner `net_dev` according to ingress port

- **Failover**
  - RoCE and user mode Eth traffic transport object (QP) port is migrated transparently in HW
  - Traditional `net_dev` I/F traffic remains associated with slave `net_dev`

- **Verbs**
  - Use transport object (QP) attribute: port affinity

- **Configuration**
  - Native Linux administration
  - RoCE Bonding is mainly auto configured

- **LACP ((802.3ad)**
  - Either handled by Linux bonding/teaming driver
  - Or in HW/FW for supporting NICs (required for many PFs to single phys port configurations)
HW BOND AND VIRTUALIZATION

eSwitch Software Model – Option II

Native OS

RDMA Device

eth0

Linux/OVS Bridge

VM2

VM3

SRIOV VM0

SRIOV VM1

rep_eth0

rep_phy0

rep_vf0

rep_vf1

PCIe PF0

eSwitch

NIC

Phys Port

PCIe VF0.0

PCIe VF0.1
HW BOND AND VIRTUALIZATION

eSwitch Software Model with Tunneling

- OVS-VX Bridge
- UDP/IP Stack
- RDMA Device
- eth0
- Linux/OVS Bridge
- rep_eth0
- rep_phy0
- rep_vf0
- rep_vf1
- VM2
- VM3
- SRIOV VM0
- SRIOV VM1
- vxlan net_device
- VNI (Key)
- PCIe PF0
- PCIe VF0.0
- PCIe VF0.1
- eSwitch
- HW Tunnel
- Phys Port
- NIC
MULTI-PCI SOCKET NIC

- Single NIC can be connected through one or more PCIe buses
- Each PCIe bus is connected through different NUMA node
- For OS, exposed as 2 or more net_device each with it’s own associated RDMA device
- Application enjoy direct device to local NUMA access
  - Using local network I/F per NUMA node
- **Boosting performance for HPC and Cloud**
  - QPI avoidance for I/O – Optimal performance
  - Enables GPU / peer direct on both slots
  - Enables Direct Data I/O (DDIO) acceleration for both sockets
MULTI-PCI SOCKET NIC
Benchmark

20% Lower Latency

TCP 300 streams average Latency

CPU % Receive

50% of CPU Overhead
MULTI-PCI SOCKET NIC
Transparency to the App

- Application use & feel – would like to work with single net I/F
- Use Linux bonding with RDMA device bonding
- For TCP/IP traffic
  - On TX, select slave according to TX queue affinity
  - On RX, use accelerated RFS to educate the NIC which slave to use per flow
- For RDMA/User mode ETH traffic select slave according to:
  - Explicit - Transport object (QP) logical port create affinity attribute
  - Or transport object creation thread CPU affinity attribute
  - QPn namespace is divided across slaves
    - On receive use QPn to slave mapping
    - From BTH or from Flow Steering action
- Don’t share HW resources (CQ, SRQ) on different CPU sockets
  - each device has it’s own HW resources
SUMMARY

- Traditional stack transport logic is managed in software (TCP/IP)
- RDMA transport logic is managed in NIC HW
- Migrating the HW managed transport object from failed port requires HW aid
  - Currently limited to phys port of the same adaptor
- Building on top of existing infrastructure provides seamless administrative and application wise configuration
  - Allows HW awareness of the configuration and failover event
- Same logic may be used for representing multiple logical devices to single phys device interface
THANK YOU

Tzahi Oved

Mellanox Technologies